

Appendix D: Error Detection

KaBOB's integrated biological model and its commitment to data provenance provide a unique opportunity for error detection when compared to other data integration systems. Errors can be detected by querying for concepts or links between concepts that should not exist. Once identified their provenance can be traced, and a solution can be identified. This capability is a critical aspect to the rule writing process and has been successfully used to identify and repair logic errors in rules. Errors in the source data can also be detected. In these cases it is often possible to adjust relevant rules can to correct for these problems. More rare are errors in a file parser causing the generation of erroneous RDF.

Disjoint Types

One straightforward thing to query for is to ensure that biomedical concepts do not exist that are of disjoint types. This can be accomplished with some targeted SPARQL queries, however, even more leverage could be gained here from the use of OWL reasoners, which is a source of future work. Things that can be queried for include ensuring entities are distinct from processes and that DNA and proteins are distinct from each other, which is shown in the following query:

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX obo: <http://purl.obolibrary.org/obo/>

SELECT ?bio WHERE {
  ?bio rdfs:subClassOf* obo:SO_0000352 . #DNA
  ?bio rdfs:subClassOf* obo:CHEBI_36080 . #protein
}
```

Identifier Collapse

Since KaBOB resolves identity across data sources by clustering sets of identifiers, it is important to detect if a set of identifiers has accidentally grown too large, causing what should be several distinct biomedical entities to collapse into a single identifier. This can be hard to detect prior to the identifier sets being generated. The ability to

31 inspect the consequences of these identifier mappings is an important advantage
32 provided by KaBOB, as opposed to systems that require query writers to manually
33 transverse these mappings themselves (as is the case with the current state of the art in
34 biomedical pipeline building, or when using a system such as Bio2RDF).

35 One mode of identifier collapse is when a data source does not uniquely map to
36 identifiers in another source, as we detected in DrugBank. These erroneous mappings
37 resulted in 300 errors where two or more DrugBank identifiers were associated with
38 the same external identifier. Since a transitive closure is computed over identifier
39 mappings it is also possible to detect distant interactions not otherwise noticeable. For
40 example, if there are mappings from source A to source B, B to source C, and then C
41 to A, it is possible that the mappings do not agree and instead of tight loops of
42 agreement being formed there are disagreements that when transitively closed cause
43 too many identifiers to be merged. If you only looked at all the mappings in and out of
44 a source those disagreements would not be seen.

45 The first naïve approach to detecting collapse is to look for sets of identifiers that
46 include multiple identifiers from the same data source. However, since many data
47 sources have redundant and deprecated identifiers, for example UniProt, this will not
48 universally hold. Instead we look at the primary identifier fields for many of the data
49 sources. Since these should all point to distinct biomedical entities, we can check for
50 sets that contain two primary identifiers from UniProt, or from DrugBank, etc. An
51 example of this kind of error checking query looks like the following:

```
52 PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
53 PREFIX obo: <http://purl.obolibrary.org/obo/>
54 PREFIX kiao: <http://kabob.ucdenver.edu/iao/>
55 PREFIX iaodrugbank: <http://kabob.ucdenver.edu/iao/drugbank/>
56
57 SELECT ?dbId1 ?dbId2 ?bio WHERE {
58
59 #look for the first DrugBank ID - primary ID field
60 ?fv1 kiao:hasTemplate
61      iaodrugbank:DrugBankDrugRecord_drugBankIdDataField1 .
```

```

62 ?fv1 obo:IAO_0000219 ?dbId1 . #denotes
63
64 ?dbId1 obo:IAO_0000219 ?bio . #denotes
65 ?dbId2 obo:IAO_0000219 ?bio . #denotes
66
67 FILTER (?dbId1 != ?dbId2) #make sure they are different
68
69 #look for the second DrugBank ID - primary ID field
70 ?fv2 obo:IAO_0000219 ?dbId2 . #denotes
71 ?fv2 kiao:hasTemplate
72     iaodrugbank:DrugBankDrugRecord_drugBankIdDataField1 .
73 }
74

```

75 From this query we were initially able to detect identifier collapse. When we

76 traced back the provenance we identified that DrugBank was non-uniquely mapping

77 to external identifiers. These were being extracted using this rule:

```

78 `{:name "drugbank-drug-exact-mapping-assertion"
79   :head ((?/dbice skos/exactMatch ?/otherice))
80   :body
81     ((?/fv0 kiao/hasTemplate
82         iaodrugbank/DrugBankDrugRecord_drugBankIdDataField1)
83      (?/fv0 obo/IAO_0000219 ?/dbice)
84      (?/record obo/has_part ?/fv0)
85
86      (?/record obo/has_part ?/externalfv)
87      (?/externalfv kiao/hasTemplate
88         iaodrugbank/DrugBankDrugRecord_externalIdentifiersDataField1)
89      (?/externalfv obo/IAO_0000219 ?/otherice))
90 }
91

```

92 That rule was updated to remove the unreliable identifiers from consideration and

93 KaBOB was able to be regenerated without these entities collapsing. The updated rule

94 looks like the following:

```

95 `{:name "drugbank-drug-exact-mapping-assertion"
96   :head ((?/dbice skos/exactMatch ?/otherice))
97   :body
98     ((?/fv0 kiao/hasTemplate
99         iaodrugbank/DrugBankDrugRecord_drugBankIdDataField1)
100      (?/fv0 obo/IAO_0000219 ?/dbice)
101      (?/record obo/has_part ?/fv0)
102
103      (?/record obo/has_part ?/externalfv)
104      (?/externalfv kiao/hasTemplate
105         iaodrugbank/DrugBankDrugRecord_externalIdentifiersDataField1)
106      (?/externalfv obo/IAO_0000219 ?/otherice))
107
108      ;;check to see if that fv is in another record
109      (:optional ((?/record2 obo/has_part ?/externalfv)
110                  (:not (= ?/record2 ?/record))))
111      (:not (:bound ?/record2)))
112 }
113

```

A query to get a list of the detectable mis-mapped identifiers and its results is in Appendix E.

Another form of identifier collapse we have seen was due to the RDF file parsers accidentally truncating RefSeq identifiers. In this case all RefSeq identifiers were accidentally written as only their first few digits. This caused the majority of the entire human proteome to collapse to a single biomedical entity in KaBOB. This was immediately detected and then resolved.

Taxon Disagreement

Since we have taxon information for all the species-specific data in KaBOB, errors in the biomedical model can also be detected by looking for when entities from differing species are connected in inappropriate ways. The first most obvious thing to check for is a single biomedical entity being part of two disjoint taxa. Note that it is perfectly acceptable for a biomedical concept to have multiple taxon assertions, for example all the parent taxon classes are necessarily entailed by a child assertion, *e.g.*, all *Homo sapiens* (NCBI Taxon 9606) are also members of *Mammalia* (NCBI Taxon 40674). A query for an entity in two disjoint taxa is the following:

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX obo: <http://purl.obolibrary.org/obo/>

SELECT ?dna ?tax1 ?tax2
WHERE {
  ?dna rdfs:subClassOf obo:SO_0000352 .                #DNA
  ?dna rdfs:subClassOf ?taxonrestriction1 .
  ?taxonrestriction1 owl:onProperty obo:RO_0002162 . #in_taxon
  ?taxonrestriction1 owl:someValuesFrom ?tax1 .

  ?dna rdfs:subClassOf ?taxonrestriction2 .
  ?taxonrestriction2 owl:onProperty obo:RO_0002162 . #in_taxon
  ?taxonrestriction2 owl:someValuesFrom ?tax2 .

  FILTER (?tax1 != ?tax2) .
  FILTER NOT EXISTS { ?tax1 rdfs:subClassOf* ?tax2 . } .
  FILTER NOT EXISTS { ?tax2 rdfs:subClassOf* ?tax1 . } .
}
```

150 The source data that KaBOB is derived from is quite consistent in this regard and
 151 only six errors are propagated into KaBOB from erroneous data. These include one
 152 protein that is confused between mouse and rat, and five proteins that are
 153 simultaneously in two sibling yeast strains. The following are those results.

```
154 obo:NCBITaxon_10090 obo:NCBITaxon_10116 (mouse rat)
155     kbio:BIO_e5838794206c4357c4fc0887ff9ef8f4
156 obo:NCBITaxon_10090 obo:NCBITaxon_10116 (yeast)
157     kbio:BIO_c8742d154afdfecae05bbf42544456f8
158     kbio:BIO_a7f26ae6f4a59610e55d080a894fb606
159     kbio:BIO_4998abaafcf5a986229945dbfaef54f1
160     kbio:BIO_19cladaabcd380dece3abeca9b4cde54
161
```

162 Entities also stand in relationship to each other, for example, gene products (*e.g.*,
 163 proteins) are coded for by genes. Taxon information can also be leveraged here to
 164 look for misalignments by checking for a gene product from one taxa having a gene
 165 from a disjoint taxa as its template. A query that looks for that is the following:

```
166 PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
167 PREFIX owl: <http://www.w3.org/2002/07/owl#>
168 PREFIX obo: <http://purl.obolibrary.org/obo/>
169
170 #retrive dna, gene-product, dna taxa, gene-product taxa
171 SELECT ?dna ?gptaxon2 ?tax1 ?tax2
172 WHERE {
173     #DNA/gene in a taxa
174     ?dna rdfs:subClassOf obo:SO_0000352 .                #DNA
175     ?dna rdfs:subClassOf ?taxonrestriction1 .
176     ?taxonrestriction1 owl:onProperty obo:RO_0002162 . #in_taxon
177     ?taxonrestriction1 owl:someValuesFrom ?tax1 .
178
179     #get the gene or gene product abstraction for that gene
180     ?dna rdfs:subClassOf* ?gorgporv .
181     ?gorgporv rdf:type kbio:GeneSpecificGorGPorVClass .
182
183     #look at all the taxa of the member gene products
184     ?gptaxon2 rdfs:subClassOf* ?gorgporv .
185     ?gptaxon2 rdfs:subClassOf ?taxonrestriction2 .
186     ?taxonrestriction2 owl:onProperty obo:RO_0002162 . #in_taxon
187     ?taxonrestriction2 owl:someValuesFrom ?tax2 .
188
189     #filter out those that are allowed
190     FILTER (?tax1 != ?tax2) .
191     FILTER NOT EXISTS { ?tax1 rdfs:subClassOf* ?tax2 . } .
192     FILTER NOT EXISTS { ?tax2 rdfs:subClassOf* ?tax1 . } .
193 }
194
```